Name: Rupesh Dhirwani	**Class/Roll No. :** D16AD-10	**Grade:**

Title of Experiment: Multi-layer perceptron algorithm to simulate XOR gate.

Objective of Experiment: The objective of this experiment is to build a Multi-layer perceptron network that can accurately simulate the XOR's gate behavior. The network should take two binary inputs as (0 or 1) and produce the corresponding binary outputs, i.e (0 or 1) that align with the XOR truth table.

Outcome of Experiment: We have successfully implemented the Multi-layer perceptron.

Problem Statement: Design and implement Multi-layer perceptron algorithm to simulate XOR gate.

Description / Theory :

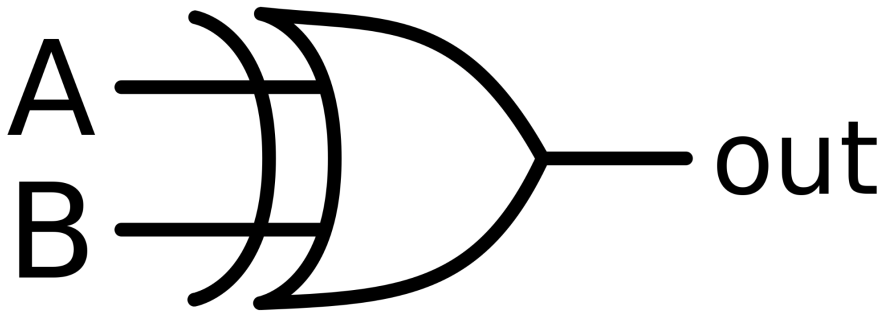
XOR GATE: XOR gate, short for "exclusive OR" gate, is a fundamental digital logic gate that performs a specific logical operation on two binary input signals (usually denoted as A and B). The XOR gate produces a binary output signal (often denoted as Y or Z) based on the following truth table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

From the truth table, the XOR gate output is 1 (true) if the number of 1s in the input is odd, and it's 0 (false) if the number of 1s in the input is even.

Which means, it produces a high output (1) only when the inputs are different from each other. XOR gates are often used in digital circuits and computer systems for various purposes, including arithmetic operations, error checking, and generating

complex logic functions. They are a fundamental building block in digital circuit design and play a crucial role in creating more complex logic operations and circuits.



Multi-layer Perceptron:

A Multi-Layer Perceptron (MLP) is a type of artificial neural network architecture that consists of multiple layers of interconnected nodes (neurons) organized in a feedforward manner. Each neuron in a layer is connected to all the neurons in the previous and subsequent layers, but there are no connections within a single layer. MLPs are a foundational concept in the field of deep learning and are used for various tasks such as classification, regression, and more complex tasks like image recognition and natural language processing.

The basic structure of an MLP includes the following layers:

1. Input Layer: This layer receives the input data, which could be a set of features from an image, text, or any other data format. Each neuron in the input layer corresponds to a specific feature.

2. Hidden Layers: These are the intermediate layers between the input and output layers. Each hidden layer consists of multiple neurons that perform computations on the input data using weighted connections and activation functions. The number of hidden layers and the number of neurons in each layer are design choices that can influence the network's performance and complexity.

3. Output Layer: The final layer produces the network's prediction or output. The number of neurons in the output layer depends on the type of problem being solved. For example, in a binary classification task, there might be one neuron representing the probability of one class and another neuron representing the probability of the other class.

The connections between neurons in different layers are associated with weights that are learned during the training process. These weights determine how information flows through the network and are adjusted iteratively to minimize the difference between the network's predictions and the actual target values using a process called backpropagation.

Activation functions are applied to the output of neurons in the hidden layers. Common activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). These functions introduce non-linearity to the network, allowing it to capture complex relationships within the data.

MLPs can be used for a wide range of tasks, and their effectiveness is greatly enhanced when combined with techniques like regularization, dropout, and optimization algorithms. However, they may struggle with handling certain types of data, such as sequences or spatial relationships, which has led to the development of more specialized architectures like recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

Program :

```
import numpy as np


def unit_step(v):
    return 1 if v >= 0 else 0


def perecep_model(x, w, b):
    v = np.dot(x, w) + b
    y = unit_step(v)
    return y
```

```
def NOT_logic_function(x):
    w_NOT = -1
    b_NOT = 0.5
    return perecep_model(x, w_NOT, b_NOT)


def AND_logic_function(x):
    w_AND = np.array([1, 1])
    b_AND = -1.5
    return perecep_model(x, w_AND, b_AND)


def OR_logic_function(x):
    w_OR = np.array([1, 1])
    b_OR = -0.5
    return perecep_model(x, w_OR, b_OR)


def XOR_logic_function(x):
    y1 = AND_logic_function(x)
    y2 = OR_logic_function(x)
    y3 = NOT_logic_function(y1)
    final_x_is = np.array([y2, y3])
    return AND_logic_function(final_x_is)
```

```
test_cases = [
    (0, 0),
    (0, 1),
    (1, 0),
    (1, 1),
]

print("\n\nResults:")
for test_case in test_cases:
    x1, x2 = test_case
    output_final = XOR_logic_function(np.array([x1, x2]))
    print(f"\tXOR({x1}, {x2}) = {output_final}")
```

OUTPUT:

```
Results:
        XOR(0, 0) = 0
        XOR(0, 1) = 1
        XOR(1, 0) = 1
        XOR(1, 1) = 0
PS C:\Users\Asus\OneDrive\Desktop\New folder>
```

Results and Discussions :

After training the multilayer perceptron on the XOR dataset, the model has been evaluated based on the accuracy of its predictions. The model performance is accessed by comparing the predicted output with the ground truth of its truth values of the XOR gate.

Also, the multi-layer perceptron algorithm proves to be capable of simulating the XOR gate successfully. Through its ability to model the non-linear relationship between the inputs and outputs, also the hidden layer in the multi-layer perceptron allows it to learn the XOR function efficiently.

The number of hidden layers in the multi-layer perceptron is a crucial hyper-parameter that influences the performance of multi-layer perceptrons. Since fewer or few number of neurons can lead to under-fitting in the program where the model is unable to learn about the complexity of XOR gate. On the other hand, too many neurons can lead to overfitting, causing the model to memorize the training data without generalizing well to the new unseen data.

Finally, through appropriate hyper-parameter tuning, the multi-layer perceptron can achieve high accuracy in simulating the XOR GATE that demonstrates the capability of Artificial Neural Networks to handle non- linearly separable problems making them a powerful tool to solve real-world problems.